

# INDUÇÃO MATEMÁTICA

RAUL H.C. LOPES

## 1. INDUÇÃO MATEMÁTICA

**1.1. Conjunto indutivamente definido.** Considere a seguinte definição, encontrada em [5].

**Definição de um conjunto indutivo.** *Um conjunto de números reais é chamado de conjunto indutivo se ele tem as seguintes propriedades:*

- (1) *O número 1 pertence ao conjunto.*
- (2) *Para todo  $x$  pertencente ao conjunto, o número  $x + 1$  também pertence ao conjunto.*

Essa definição apresenta um *método para computar* números reais a partir de um elemento inicial: o número 1. Note que essa definição limita-se a estabelecer como encontrar elementos de **um** conjunto indutivo de reais (os inteiros positivos): ela nem tenta cobrir todo o conjunto de reais e nem mesmo determinar exatamente todos os elementos do conjunto indutivo em questão.

Veja agora a seguinte definição retirada da mesma fonte.

**Definição de inteiros positivos.** *Um número real é chamado inteiro positivo se ele pertence a todo conjunto indutivo.*

A definição acima, claramente referindo-se e complementando a anterior, tentar definir exatamente o conjunto dos inteiros positivos estabelecendo que ele é a interseção de todos os conjuntos possíveis de se obter, usando a primeira definição: ou em outras palavras, ele é o menor conjunto que contém todos os elementos ditados pelos 2 itens da primeira definição.

Além de estabelecer um “método para enumerar” todos os elementos do conjunto de inteiros positivos, essas duas definições “induzem”:

- i um método para testar se qualquer  $z$  é um inteiro positivo: verificamos se  $z$  é o elemento inicial do conjunto definido ou se ele pode ser reescrito como  $x + 1$ , para algum  $x$  pertencente ao conjunto definido.
- ii um método de prova de propriedades desse conjunto. Para provar a propriedade  $P_n$ , para todo  $n \geq n_0$ :
  - 1 Provamos que  $P$  é válido para o  $n_0$ :  $n_0$  é a agora o elemento inicial de uma subsequência de interesse;

2 Provamos que, quando  $k \geq n_0$ ,  $P.k$  implica  $P.(k+1)$ , ou seja para qualquer elemento da seqüência se  $P$  vale para esse elemento, então  $P$  vale para o seu sucessor.

**Exemplo 1.** Prove que para  $n$  pertencente ao conjunto de inteiros acima definido

$$\forall n : n \geq 4 : n^2 \leq 2^n$$

*Demonstração.*  $P.n$  a ser provado é  $n^2 \leq 2^n$  e que deve ser válido com  $n \geq 4$ . Seguindo roteiro de prova traçado acima:

- 1 quando  $n = 4$ ,  $P.n$  reduz-se a  $4^2 \leq 2^4$ , que é trivialmente verificado.
- 2 para provar que  $P.k$  implica  $P.(k+1)$ , começamos com  $P.k$

$$\begin{aligned} P.k &= k^2 \leq 2^k \\ &= \langle \text{multiplicando por } 2 \rangle \\ &2.k^2 \leq 2.2^k \\ &= \langle \text{porque } (k+1)^2 \leq 2k^2, \text{ para } k \geq 3 \rangle \\ &(k+1)^2 \leq 2^{k+1} \\ &= P.(k+1) \end{aligned}$$

□

**1.2. Provas por indução.** Obviamente a construção de conjuntos de inteiros não precisa ter como elemento inicial o número 1. Assuma agora que  $\mathbb{N}$  é o conjunto dos inteiros não negativos: na definição acima, assumo que o número 0 é o elemento inicial.

**Exercício 1.** Prove por indução:

- (1)  $(\sum i : 1 \leq i \leq n : )i = n(n+1)/2$
- (2)  $(\sum i : 0 \leq i < n : 2^i) = 2^n - 1$
- (3)  $(\sum i : 0 \leq i < n : 3^i) = (3^n - 1)/2$
- (4)  $(\sum i : 1 \leq i \leq n : i^2) = n(n+1)(2n+1)/6$
- (5)  $(\sum i : 0 \leq i \leq n : i.2^i) = (n-1).2^{n+1} + 2$
- (6)  $(\sum i : 0 \leq i \leq n : (2i+1)^2) = (n+1)(2n+1)(2n+3)/3$
- (7)  $(\sum i : 0 \leq i \leq n : i(i+1)(i+2)) = n(n+1)(n+2)(n+3)/4$
- (8)  $\forall n, a : n \geq 1 \wedge a \neq 1 : (\sum i : 0 \leq i < n : a^i) = (1 - a^n)/(1 - a)$
- (9)  $\forall n : n \geq 1 : (\sum i : 1 \leq i \leq n : 1/(i.(i+1))) = n/(n+1)$
- (10)  $\forall n : n \geq 3 : n+1 < 2^n$
- (11)  $\forall n : n \geq 4 : n^2 \leq 2^n$
- (12)  $\forall n : n \geq 7 : 3^n < n!$

*Demonstração.*  $P.n$  a ser provado é  $3^n < n!$  e que deve ser válido com  $n \geq 7$ . Seguindo roteiro de prova traçado acima:

1 quando  $n = 7$ ,  $P.n$  reduz-se a  $3^7 < 7!$ , que é trivialmente verificado.

2 para provar que  $P.k$  implica  $P.(k+1)$ , começamos com  $P.k$

$$P.k = 3^k \leq k!$$

$$= \langle \text{multiplicando por } (k+1), \text{ dado que } k \geq 7 \rangle$$

$$(k+1).3^k \leq (k+1).k!$$

$$= 3^{k+1} + 3^k \leq (k+1)!$$

$$= \langle \text{porque } 3^{k+1} \leq 3^{k+1} + 3^k \text{ quando } k \geq 7 \rangle$$

$$3^{k+1} \leq (k+1)!$$

$$= P.(k+1)$$

□

**1.3. Ouro de tolo?** Já dá para conjecturar: o método de prova acima, o chamado *Princípio da Indução*, deve ser aplicável à prova de propriedades de qualquer conjunto contável. Então... Dado o conjunto de humanos, que nem mesmo é infinito, ficamos tentados a realizar aplicações mais interessantes.

**Exercício 2.** *Prove que todas as garotas loiras têm olhos claros.*

Para obter a prova acima, você pode:

- (1) Esquecer o Princípio da Indução e partir para o experimento. Que tal na sua turma de *Estruturas de Dados*? E' claro que isso demanda definir:
  - i O que são garotas? *Ok! Isso pode não ser politicamente correto!*
  - ii O que são olhos claros?
- (2) Tentar Princípio da Indução, mas quem são  $P.k$  e  $P.(k+1)$ ? Aliás, esse é o ponto fundamental de toda a prova por indução: indentificar  $P.k$ , a *hipótese de indução*, e realizar o *passo indutivo* para obter  $P.(k+1)$ .

Ok! De volta ao reino de  $\mathbb{N}$ ...

Veja o seguinte “teorema” e sua “prova”, retirados de [3].

**Teorema 1.** *Dados inteiros positivos  $a$  e  $n$ ,  $a^{n-1} = 1$ .*

**Prova Suspeita.**

1 Para  $n = 1$ ,  $a^{1-1} = a^0 = 1$ .

2 Para  $n = k+1$ ,

$$a^{(k+1)-1} = a^k = \frac{a^{k-1} \cdot a^{k-1}}{a^{(k-1)-1}} = \frac{1 \cdot 1}{1} = 1$$

O que está errado? Talvez o fato de que para se obter a prova para  $n = k + 1$ , foram usados como hipóteses o fato de que a assertiva seria válida para  $n = k$  e  $n = k - 1$ ? Não! Na verdade, como mostrado na próxima seção, esse é um raciocínio válido.

1.4. **Indução forte.** Considere o seguinte problema.<sup>1</sup>

**Exemplo 2.** Para qualquer número  $n \geq 8$ , é possível obter a partir de uma soma de fatores todos iguais a 3 ou 5.

**A caminho da prova.**

- (1) Para  $n = 8$ ,  $8 = 5 + 3$ .
- (2) Provar  $P.(k + 1)$ , assumindo apenas  $P.k$  é mais difícil. Tente provar que  $P.(k + 1)$ , assumindo que  $P.k$  para  $8 \leq k \leq n$ .

**Teorema 1.** Seja  $P$  uma propriedade sobre  $\mathbb{N}$  e assumamos foram provados:

- i  $P.0$ ;
- ii  $\forall n : n \in \mathbb{N} : (\forall m \in \mathbb{N} : m \leq n : P.m) \Rightarrow P.(n + 1)$

Prove que  $\forall n : n \in \mathbb{N} : P.n$ .

*Demonstração.* Seja  $Q.n = \forall m \in \mathbb{N} : m \leq n : P.m$ .

Por indução, é fácil provar que  $\forall n : n \in \mathbb{N} : Q.n$ .

- 1  $Q.0 = \forall m \in \mathbb{N} : m \leq 0 : P.m = P.0$ , que é verdadeiro.
- 2 Assumindo que  $Q.n = \forall m \in \mathbb{N} : m \leq n : P.m$  é verdadeiro, pode-se deduzir  $P.(n + 1)$ , pelo segundo dos fatos conhecidos de  $P$ . Mas,

$$\begin{aligned} Q.n \wedge P.(n + 1) &= (\forall m \in \mathbb{N} : m \leq n : P.m) \wedge P.(n + 1) \\ &= \forall m \in \mathbb{N} : m \leq n + 1 : P.m \\ &= Q.(n + 1) \end{aligned}$$

□

A prova acima estabelece o chamado **Princípio da Indução Forte** como consequência do Princípio da Indução. O Princípio da Indução Forte ajuda na hora de resolver o problema apresentado no exemplo 2. Após a prova para o caso básico, quando  $n = 8$ , a solução prosseguiria em obter prova para o caso em que  $n = 9$  (soma de fatores iguais a 3) e provar que, assumindo que qualquer  $k$  ( $10 \leq k \leq n$ ) pode ser escrito como soma de fatores iguais a 3 ou 5,  $k + 1$  também pode ser escrito da mesma forma. Tente!

Considere a seguinte série, conhecida como série de Fibonacci<sup>2</sup>

<sup>1</sup>Retirado das notas de aula de Tom Leighton, do curso de Mathematics for Computer Science, MIT, Setembro/97.

<sup>2</sup>Veja em [2] e [3] história, fatos e folclore em torno do números de Fibonacci.

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \text{ para } n > 1$$

**Exercício 3.** Use o Princípio da Indução Forte para provar que  $F_n < 2^n$ .

Por outro lado, veja o seguinte exemplo interessante.

**Exemplo 3.** Todos os números da série de Fibonacci são pares.

**Prova Suspeita.** Por indução forte.

- 1 Quando  $n = 0$ ,  $F_n = 0$ , que é par.
- 2 Assumindo que para todo  $0 \leq k \leq n$ ,  $F_k$  é par,  $F_{n+1} = F_n + F_{n-1}$ , é par porque  $F_n$  e  $F_{n-1}$  também são, por hipótese de indução.

Humm... Mas, 1, 3, 5 todos estão na série de Fibonacci e são ímpares. Bem, talvez sejam apenas uns poucos, uma minoria. Basta disfarçar e fingir que não estão lá. Ou... “esquece tudo: essa tal de Ciência da Computação não funciona porque o fundamento matemático não é confiável.” Bem, e quem disse que indução é fundamento de alguma coisa?

### 1.5. Classe indutiva.

**Definição 1.**  $\mathfrak{X}$  é uma classe indutiva se:

- (i) seus elementos são gerados a partir de uma das seguintes operações:
  1. existe uma especificação inicial  $\mathcal{B}$  de um conjunto de elementos que pertencem a  $\mathfrak{X}$  (chamados elementos iniciais);
  2. existe um conjunto  $\mathcal{M}$  de operações que aplicadas a elementos de  $\mathfrak{X}$  produzem novos elementos de  $\mathfrak{X}$ .
- (ii) Todos elemento de  $\mathfrak{X}$  pode ser obtido começando com um ou mais elementos de  $\mathcal{B}$  e aplicando a cada passo uma operação de  $\mathcal{M}$  a elementos obtidos em algum passo anterior ou obtidos de  $\mathcal{B}$ .

É comum dizer que  $\mathcal{X}$  é a menor classe (ou o fecho universal da classe), cujos elementos iniciais são dados por  $\mathcal{B}$  e que tem novos elementos construídos por aplicações repetidas de  $\mathcal{M}$ .<sup>3</sup>

**Exemplo 4.** A série de Fibonacci é uma classe indutiva, cujos elementos iniciais são  $F_0 = 0$  e  $F_1 = 1$ . Qualquer  $F_n$ , para  $n > 1$ , é obtido da soma de  $F_{n-1}$  e  $F_{n-2}$ .

<sup>3</sup>Veja capítulos iniciais [1] sobre conceitos de classe indutiva, algoritmo, e questões definidas e semi-definidas.

O Princípio da Indução fica mais claro quando aplicado ao conceito de classe indutiva.

**Definição 2.** *Uma propriedade  $P$  é válida para todos os elementos de uma classe indutiva  $\mathfrak{X}$  se:*

- (i)  *$P$  é válida para todos os elementos iniciais de  $\mathfrak{X}$  ( $P$  é válida para os elementos gerados por  $\mathcal{B}$ ).*
- (ii) *Qualquer operação  $\mathcal{M}$  (que permite gerar elementos de  $\mathfrak{X}$  a partir de elementos já conhecidos) preserva a validade de  $P$ .*

**Exemplo 5.** *Uma propriedade é válida para classe da série de Fibonacci, definida em 4 se ela for válida para seus elementos iniciais: 0 e 1. Logo, a propriedade de que todos os elementos da classe são pares, do exemplo 3, não é válida.*

**Exercício 4.** *Reveja sua prova do exercício 3 à luz do conceito de classe indutiva.*

**Exercício 5.** *Seja  $\phi = (1 + \sqrt{5})/2$ . Prove que*

$$\phi^{n-2} \leq F_n \leq \phi^{n-1}$$

## 2. ELEMENTO MÍNIMO

**Teorema 2.** *Qualquer  $M \subseteq \mathbb{N}$ , tal que  $M$  não é vazio, tem um elemento mínimo.*

*Demonstração.* Assumindo que  $M \subseteq \mathbb{N}$  e que  $M$  não tem elemento mínimo, chega-se à contradição de que  $M$  é vazio. Por Princípio da Indução Forte, prova-se que  $\forall n : n \in \mathbb{N} : n \notin M$ .

- (1) Para  $n = 0$ ,  $n \notin M$  ou  $n$  seria mínimo, pois  $M \subseteq \mathbb{N}$  e 0 é mínimo em  $\mathbb{N}$ .
- (2) Assumindo-se que  $\forall k \in \mathbb{N} : k \notin M$ , conclui-se que  $k+1 \notin M$ , ou  $k+1$  seria mínimo de  $M$ .  
Logo  $M$  é vazio, ou  $M$  tem mínimo.

□

## 3. INDUÇÃO E COMPUTAÇÃO

**3.1. Tipos como conjuntos indutivos.** Tipos de dados são representações em alguma linguagem de programação de classes indutivas. Na verdade, qualquer computação pode ser vista como a atividade de enumerar os elementos de uma classe indutiva.

**Exemplo 6.** *Considere a classe **Natnum** como a menor classe que contém:*

- (i)  $Z$ , como único elemento inicial;
- (ii)  $(Sx)$ , quando  $x$  já pertence a **Natnum**.

3.2. **Admissibilidade de funções.** Uma representação em Haskell para a classe **Natnum** seria:

```
data Natnum = Z | (S Natnum)
```

Um exemplo de função seria:

```
itsum m Z = m
itsum m (S n) = itsum (S m) n
```

A função **itsum** acima definida permite introduzir dois conceitos importantes:

1. *Admissibilidade da função:* a função acima será dita admissível porque existe uma medida de complexidade dos seus argumentos que é sempre decrescente em qualquer chamada recursiva da mesma. Nessa função, o segundo argumento da chamada recursiva é sempre mais simples que o segundo argumento da chamada que a precede.  
Por exemplo, assuma um mapeamento  $f : \text{Natnum} \rightarrow \mathbb{N}$  que associa:

$$\begin{aligned} f.(m, Z) &= 0 \\ f.(m, (Sn)) &= f.n + 1 \end{aligned}$$

Claramente  $f$  é uma medida de complexidade para os argumentos de chamada de **itsum** decresce a cada chamada recursiva, formando um cadeia finita decrescente, com mínimo em  $(m, Z)$ .

2. Diferentemente da função **recsum** abaixo, **itsum** pode ser avaliada por um processo de repetição substituições simples da chamada original pela chamada recursiva.

```
recsum m Z = m
recsum m (S n) = S(recsum m n)
```

A computação de **recsum** demanda que uma memória seja mantida das sucessivas chamadas recursivas para os **S** sejam adicionados ao resultado final da chamada não recursiva.

Uma função interessante de se definir para a aritmética de Peano é a de subtracao. Veja as seguintes definições.

```

-- pred: predecessor de um Natnum
pred Z = Z
pred (S m) = m

-- recsub: subtracao recursiva dependente de memoria auxiliar

recsub m Z = m
recsub m (S n) = pred(recsub m n)

```

Nenhuma das definições concorda inteiramente com a aritmética de Peano: nesse sistema formal, zero não tem predecessor. Na definição acima, o objeto **Z**, introduzido para representar zero, tem como predecessor, via função **pred**, o próprio **Z**. Isso afeta diretamente a operação de subtração: a diferença de  $m$  e  $n$  quando o segundo é maior fica igual a zero.

Uma definição alternativa para a diferença de dois **Natnum**, é dada pela função iterativa **itsub**.

```

itsub (m,Z) = m
itsub (Z,n) = Z
itsub ((S m),(S n)) = itsub (m,n)

```

Antes de mais nada, note que **itsub** é admissível.

**Exercício 6.** *Prove que*

$$itsub(m, (Sn)) = pred(itsub(m, n))$$

*Use indução sobre  $n$  (segundo argumento do  $(m, n)$  da chamada recursiva.) No passo indutiva, considere dois casos: quando  $m$  é **Z** e quando  $m$  é **(S u)** para algum  $u$ .*

**3.3. Generalizando indução.** É importante observar que a medi-  
dade complexidade de argumentos de uma função não precisa se re-  
stringir a um único argumento. Como exposto em [2] (ver também [4],  
página 20, exercício 15) podemos generalizar todo o conceito de indução  
matemática e de admissibilidade funções se, dado um conjunto  $S$  (de  
inteiros, **Natnum**, tuplas, etc), pudermos estabelecer um relação  $\prec$  tal  
que:

- toda a cadeia em  $S$  tem um mínimo  $x$ : para todo  $y \in S$  existe um mínimo  $x$  tal que  $x \prec x_1 \prec x_2 \prec \dots \prec y$ .
- Se, para  $x, y, z \in S$ ,  $x \prec y$  e  $y \prec z$ , então  $x \prec z$ .

Nessa situação  $S$  é dito *bem fundado*.



Uma função recursiva é admissível se os argumentos das suas chamadas recursivas admitem algum relação de precedência que determina um relação bem fundada.

```
gcd m n = if m < n
          then gcd m (n-m)
          else if m > n
               then gcd (m-n) n
          else m
```

A relação abaixo

$$(m_0, n_0) \prec (m_1, n_1) \text{ quando } m_0 < m_1$$

$$(m_0, n_0) \prec (m_1, n_1) \text{ quando } (m_0 = m_1) \wedge (n_0 < n_1)$$

e o fato de que nas chamadas recursivas de **gcd** os argumentos de chamada são sempre maiores ou iguais do zero pode ser usada para provar que a definição de **gcd** é admissível (sempre termina.)

Por outro lado, usando indução forte, a relação  $\prec$  acima e o fato de que o maior divisor de  $m$  e  $n$  é também o maior divisor de  $m$  e  $m - n$  (quando  $m$  é maior do que  $n$ ), permite provar que a função acima calcula corretamente o maior divisor comum de dois inteiros positivos.

### 3.4. Tipos abstratos de dados.

**Exercício 7.** *Crie em Haskell um tipo de dados para representar seu indutivo da questão 6.*

**Exercício 8.** *Crie em Haskell, ao menos, as seguintes funções sobre o tipo da questão 7, defina e prove condições de correção para as mesmas:*

- (1) *Funções recursiva e iterativa para cálculo de:*
  - soma de dois objetos;
  - diferença de dois objetos;
  - produto de dois objetos;
  - divisão de dois objetos;
  - resto da divisão inteira de dois objetos.
- (2) *Funções implementar os relações lógicas, apresentando condições de correção:*
  - (a) *igualdade;*
  - (b) *desigualdade;*
  - (c) *menor que;*
  - (d) *maior que;*

**Exemplo 7.** *Dadas as definições de mult, recdiv e itmod, é possível provar que*

$$m = \text{recsum}(\text{multq}(Sn))r,$$

quando

$$\begin{aligned} q &= \text{recdiv}m(Sn) \\ r &= \text{itmod}m(Sn) \end{aligned}$$

o que corresponde ao teorema  $m = q * n + r$ , com  $n$  maior do zero.  
Para maior clareza, as seguintes abreviações serão usadas:

$m + n$  em lugar de **recsum**  $m$   $n$ ;  
 $m - n$  em lugar de **itsub**  $(m, n)$ ;  
 $m * n$  em lugar de **mult**  $m$   $n$ ;  
 $m/n$  em lugar de **recdiv**  $m$   $n$ ;  
 $m \% n$  em lugar de **itmod**  $m$   $n$ .

*Demonstração.* Por indução forte sobre  $m$ .

1.  $m = Z$ .

$$\begin{aligned} m &= ((m/(Sn)) * (Sn)) + (m \% (Sn)) \\ &= \langle \text{substituindo } m \rangle \\ &\quad ((Z/(Sn)) * (Sn)) + (Z \% (Sn)) \\ &= \langle \text{definições de recdiv, itmod} \rangle \quad (Z * (Sn)) + Z \\ &= \langle \text{definição de mult} \rangle \quad Z + Z \\ &= \langle \text{definição de recsum} \rangle \\ &\quad Z \end{aligned}$$

2. Assumindo que a propriedade é verdadeira para todo  $k$  menor ou igual a  $m$ , provamos que a mesma vale para  $k = (Sm)$ . Há dois casos a considerar:

(i)  $lt(Sm)(Sn)$ , ou seja,  $(Sm)$  menor do que  $(Sn)$ .

$$\begin{aligned} (Sm) &= (((Sm)/(Sn)) * (Sn)) + ((Sm) \% (Sn)) \\ &= \langle \text{definições de recdiv e itmod} \rangle \\ &\quad (Z * (Sn)) + (Sm) \\ &= \langle \text{definição de mult} \rangle \\ &\quad Z + (Sm) \\ &= \langle \text{definição de recsum} \rangle \\ &\quad (Sm) \end{aligned}$$

$$\begin{aligned}
& \text{(ii) quando } (Sm) \text{ é maior ou igual a } (Sn). \\
(Sm) &= (((Sm)/(Sn)) * (Sn)) + ((Sm)\% (Sn)) \\
&= \langle \text{definições de } recdiv \text{ e } itmod \rangle \\
&\quad (S(((Sm) - (Sn))/(Sn)) * (Sn)) + (((Sm) - (Sn))\% (Sn)) \\
&= \langle \text{definição de } mult \rangle \\
&\quad (((((Sm) - (Sn))/(Sn)) * (Sn)) + (Sn) + (((Sm) - (Sn))\% (Sn))) \\
&= \langle \text{comutatividade e associatividade de } recsum \rangle \\
&\quad ((((((Sm) - (Sn))/(Sn)) * (Sn)) + (((Sm) - (Sn))\% (Sn)))) + (Sn) \\
&= \langle \text{hipótese de indução} \rangle \\
&\quad ((Sm) - (Sn)) + (Sn) \\
&= (Sm)
\end{aligned}$$

□

**Exercício 9.** Repita o exercício anterior em um provador de teoremas (e.g. Isabelle, PVS), formalizando as provas correspondentes.

## REFERÊNCIAS

- [1] Haskell B. Curry, *Foundations of mathematical logic*, Dover Publications, Inc., 1977.
- [2] David Gries and Fred B. Schneider, *A logical approach to discrete mathematics*, Springer-Verlag, 1993.
- [3] Donald E. Knuth, *Fundamental algorithms*, third ed., The Art of Computer Programming, vol. I, Addison-Wesley, 1998.
- [4] ———, *Sorting and searching*, The Art of Computer Programming, vol. III, Addison-Wesley, 1998.
- [5] Tom M. Apostol, *Calculus: One-variable calculus, with an introduction to linear algebra*, second edition ed., vol. I, John Wiley & Sons, Inc., 1967.